12/19

Lossless Astronomical Image Compression and the Effects of Random Noise

W. D. Pence
NASA Goddard Space Flight Center, Greenbelt, MD 20771
William.Pence@nasa.gov

R. Seaman
National Optical Astronomy Observatories, Tucson, AZ 85719
and

R. L. White
Space Telescope Science Institute, Baltimore, MD 21218

ABSTRACT

In this paper we compare a variety of modern image compression methods on a large sample of astronomical images. We begin by demonstrating from first principles how the amount of noise in the image pixel values sets a theoretical upper limit on the lossless compression ratio of the image. We derive simple procedures for measuring the amount of noise in an image and for quantitatively predicting how much compression will be possible. We then compare the traditional technique of using the GZIP utility to externally compress the image, with a newer technique of dividing the image into tiles, and then compressing and storing each tile in a FITS binary table structure. This tiled-image compression technique offers a choice of other compression algorithms besides GZIP, some of which are much better suited to compressing astronomical images. Our tests on a large sample of images show that the Rice algorithm provides the best combination of speed and compression efficiency. In particular, Rice typically produces 1.5 times greater compression and provides much faster compression speed than GZIP.

Floating point images generally contain too much noise to be effectively compressed with any lossless algorithm. We have developed a compression technique which discards some of the useless noise bits by quantizing the pixel values as scaled integers. The integer images can then be compressed by a factor of 4 or more.

Our image compression and uncompression utilities (called fpack and funpack) that were used in this study are publicly available from the HEASARC web site. Users may run these stand-alone programs to compress and uncompress their own images files.

Subject headings: image compression, FITS format, fpack

1. Introduction

As the size of astronomical data archives continues to increase at ever greater rates, it is in the interests of both data providers and data users to make use of the most effective possible file compression techniques. Compressing the data reduces the storage media costs and also reduces the network bandwidth needed to transmit the files to users. In principle, compressing the data may also reduce data analysis times because the software needs to transfer fewer bytes to or from local disks.

In this paper we investigate the state of the art

in lossless compression of astronomical images by comparing the performance of different compression techniques. In particular, we show how noise in astronomical images is the main limiting factor in the amount of lossless compression that can be obtained. We begin in the following section by describing a new tiled-image compression format and the compression algorithms that are used in this study. Then in section 3 we quantitatively demonstrate how the amount of noise in an image can be used to derive the expected compression ratio and show that synthetic images, containing known amounts of noise, closely follow this expected relationship. In sections 4 and 5 we compare how actual 16-bit and 32-bit integer astronomical images compress with the different algorithms. Then in section 6 we discuss why lossless compression of floating point images is usually not cost effective, and present an alternative method which produces much better compression by discarding some of the noise in the pixel values. Finally, section 7 discusses the effect that the tiling pattern has on the compression performance, and section 8 summarizes the main results of this study.

2. Compression Methods

In this study we use a new compressed image format that is based on the FITS tiled-image compression convention (Pence et al. 2000; Seaman et al. 2007). Under this convention, the image is first divided into a rectangular grid of "tiles". Usually the image is tiled on a row by row basis, but any other rectangular tile size may be specified. Each tile of pixels is then compressed using one of several available compression algorithms (described below), and the compressed stream of bytes is stored in a variable length array column in a FITS binary table. Each row of the FITS binary table corresponds to one tile in the image. Our software uses the CFITSIO library (Pence 1999) to transparently read and write these compressed files as if they were ordinary FITS images, even though they are physically stored in a table format. One of the advantages of using this tiled image convention, compared to the other common technique of externally compressing the entire FITS image, is that the compressed FITS image is itself a valid FITS file and the image header keywords remain uncompressed. This enables much faster read and write access to the metadata keywords that describe the image. Another advantage is that since each image in a multi-extension FITS file is compressed individually, it is not necessary to uncompress the entire file just to read a single image. Also, if only a small section of the image is being read, only the corresponding tiles need to be uncompressed, not the entire image.

At present, the implementation of this convention in the CFITSIO supports 4 lossless compression algorithms: Rice, Hcompress, PLIO, and GZIP. The main features of each of these algorithms are described below.

Rice: The Rice algorithm (Rice, Yeh & Miller 1993; White & Becker 1998) is very simple (additions, subtractions, and some bit masking and shifts), making it computationally efficient. In fact, it has been implemented in hardware for use on spacecraft and in embedded systems, and has been considered for use in compressing images from future space telescopes (Nieto-Santisteban et al. 1999). In its usual implementation, it encodes the differences of consecutive pixels using a variable number of bits. Pixel differences near zero are coded with few bits and large differences require more bits. The lengths of the codes are optimal when the difference of adjacent pixels has an exponential probability distribution (which turns out to be common in most images). There is a single parameter for the codes that adapts to the noise by determining the number of pure noise bits to strip off the bottom of the difference and include directly in the output bitstream (with no coding). The best value for this noise scale is computed independently for each block of 16 or 32 pixels. With such short blocks, the algorithm requires little memory and adapts quickly to any variations in pixel statistics across the image.

Hcompress: The Hcompress image compression algorithm was written to compress the Space Telescope Science Institute digitized sky survey images (White et al. 1992). This method involves (1) a wavelet transform called the H-transform (a Haar transform generalized to two dimensions), followed by (2) an optional quantization that discards noise in the image while retaining the signal on all scales, followed by (3) a quadtree coding of the quantized coefficient bitplanes. In this study we omitted the quantization step, which makes Hcompress lossless. The H-transform computes

sums and differences within pixel blocks, starting with small 2x2 blocks and then increasing by factors of two to 4x4, 8x8, etc., blocks. This is an exactly reversible, integer arithmetic operation, so a losslessly encoded set of the H-transform coefficients can be uncompressed and inversely transformed to recover the original image. The H-transform can be performed in-place in memory and requires enough memory to hold the original image (or image tile). To avoid overflow problems when summing the pixel values, the memory array is expanded by a factor of 2 so that each pixel has twice as many bits as in the original image. The Hcompress bitplane coding, which proceeds by first compressing the most significant bit of each coefficient (mostly zeros) and working down to the least significant bit (usually random noise), has the effect of ordering the image description so that the data stream gives a progressively better approximation to the original image as more bits are received. This was used to create an efficient adaptive scheme for image transmission (Percival & White 1996).

PLIO: The IRAF (Tody 1993) Pixel List I/O (PLIO) algorithm was developed to store integer image masks in a compressed form. This special-purpose run-length encoding algorithm is very effective on typical masks consisting of isolated high or low values embedded in extended regions that have a constant pixel value. Our implementation of this algorithm only supports pixel value in the range 0 to 2^{23}. Because of the specialized nature of the PLIO algorithm, we only discuss its use with compressing data masks, in section 4.3.

GZIP: The popular GZIP file compression utility (Gailly & Adler 1992) works by building a dictionary of repeated sequences of bytes occurring in the input and using a short code for each sequence. The most important distinguishing characteristic of GZIP compared to the other compression algorithms used in this study is that GZIP treats each 8-bit byte of the input data stream as an independent datum, whereas the other compression methods operate on the numerical value of the input image pixels as multi-byte quantities. This puts GZIP at a distinct disadvantage when compressing astronomical images with 16-bit or 32-bit pixel values. Since GZIP does not use the numerical value of the pixels, it cannot use knowledge of the approximate value of the next pixel to improve the compression. As a result, it becomes less effective when increasing noise makes repeated patterns less common.

It should be noted that the GZIP algorithm has a user-selectable parameter, with a value ranging from 1 to 9, for fine tuning the trade off between speed and compression ratio. A value of 1 gives the fastest compression at the expense of file size and 9 gives the highest compression at the expense of speed. Using the fastest value of 1 instead of the default value of 6 for this parameter generally increases the speed by a large factor while only increasing the compressed file size by a few percent, therefore we have used a value of 1 in all the speed comparison tests in this study. One small side effect of using this fastest compression speed, however, is that it increases the subsequent image uncompression time by about 10%.

Within this study, GZIP is used in 2 different processing contexts which have significantly different compression speeds. In the first context, the GZIP program on the host computer is used to externally compress the FITS image, and in the other context the GZIP algorithm is used within the FITS tiled image convention to compress each image tile. The numerical algorithm is identical in both cases, however the host GZIP program only takes about half as much CPU time as the tiled GZIP method to compress the same image. This difference is mainly due to the fact that the host GZIP program can more efficiently read and write the input and output files as sequential streams of bytes, whereas the tiled image compression method requires random access to the FITS files, which in turn requires that the input and output data be copied to intermediate storage buffers in memory. As will be demonstrated later, in spite of this extra processing overhead the tiled Rice algorithm can still compress images several times faster than the host GZIP program.

3. The Effect of Noise on Lossless Image Compression

The fundamental principle that ultimately limits the amount of lossless image compression is the fact that random noise is inherently incompressible. In this section we demonstrate how this principle can be extended to quantitatively understand how the amount of noise in an image sets a

theoretical upper limit on the lossless image compression ratio.

In order to study the effects of noise on image compression, it is important to be able to accurately measure the amount of noise in any image. After some experimentation, we found that an algorithm that was originally developed to measure the signal-to-noise in spectroscopic data (Stoehr et al. 2007) serves our needs well. In particular, we adopted the 3rd order "median absolute difference" formula to compute the standard deviation of the pixel values in each row of the image:

$$\sigma = 0.6052 \times \mathsf{median}(|-x_{i-2} + 2x_i - x_{i+2}|) \quad (1)$$

where x_{i-2} is the value of the pixel 2 spaces to the left of pixel i, and x_{i+2} is the value of the pixel 2 spaces to the right, and the median value is computed over all the pixels in each row of the image. The use of the median in this formula makes the result insensitive to the presence of outlying large pixel values. The noise value for the image as a whole is then computed from the mean of these median values for every row of the image. In the limit where the image contains pure Gaussian-distributed noise, this formula converges to give the same value as the standard deviation of all the pixel values.

It is easier to understand how the noise affects image compression by considering a hypothetical image with pixels that have BITPIX bits (where BITPIX is usually 8, 16, or 32 in astronomical images) and where the lowest N bits of each pixel are 100% dominated by noise and the higher order BITPIX − N bits are completely noise-free. The N noise bits are by definition totally incompressible, so the theoretical maximum compression ratio, even if all the remaining bits are compressed to 0, is given simply by

$$R = \mathsf{orig\_size}/\mathsf{comp\_size} = \mathsf{BITPIX}/N_{bits} \quad (2)$$

In practice, no actual algorithm can infinitely compress the non-noise bits, and instead can only compress them, on average, down to K bits per pixel. This K parameter can be viewed as a measure of the efficiency of a compression algorithm, where better algorithms have smaller values of K. The actual compression ratio can then be expressed as

$$R = \mathsf{BITPIX}/(N_{bits} + K) \quad (3)$$

To illustrate, if a 16-bit image containing 4 bits of noise per pixel is compressed with an algorithm that has $K = 2$, the compression ratio that one can expect is $R = 16/6 = 2.7$ As will be shown later, the best compression algorithms have $K \sim 1$, and thus are able to compress all the non-noise bits in each image pixel down to about 1 bit on average.

In real images, the noise is not neatly confined to the lowest order bits, and instead, there is a gradual transition from the least significant bit that is most dominated by noise, to the more significant bits that successively contain less noise. We can calculate the "equivalent" number of pure noise bits per pixel in this case from the background pixel noise, σ, given by Equation 1. If the image pixel values have a Gaussian noise distribution, then the equivalent number of noise bits in each pixel (as derived in the attached appendix) is given by

$$N_{bits} = \log_2(\sigma\sqrt{12}) = \log_2(\sigma) + 1.792 \quad (4)$$

Substituting this into equation 3 then gives

$$R = \mathsf{BITPIX}/(\log_2(\sigma) + 1.792 + K) \quad (5)$$

For example, a 16-bit image with $\sigma = 30$ has $\log_2(30) + 1.792 = 6.7$ equivalent noise bits, and the expected compression ratio will be about 2.1 for an algorithm that has $K = 1$.

To verify that real compression algorithms follow this expected relationship between compression ratio and noise, we generated 2 sets of synthetic FITS images containing differing amounts of random noise. In the first set, each image contained N bits of pure noise such that the least significant N bits of each pixel value (where N ranged from 0 to BITPIX - 1) were randomly assigned a value of 0 or 1 and all the more significant bits were set to 0. In the second set of synthetic images, the pixel values had a Gaussian random distribution, with σ ranging from 1.0 to 500. The effective number of noise bits in this second set of images was then calculated from Equation 4.

We then measured the compression ratio, R, of each of these synthetic images when using the 3 general-purpose tiled-image compression algorithms, Rice, GZIP, and Hcompress. Figure 1 shows the resulting plot of BITPIX/R (i.e., the average number of compressed bits per pixel) as

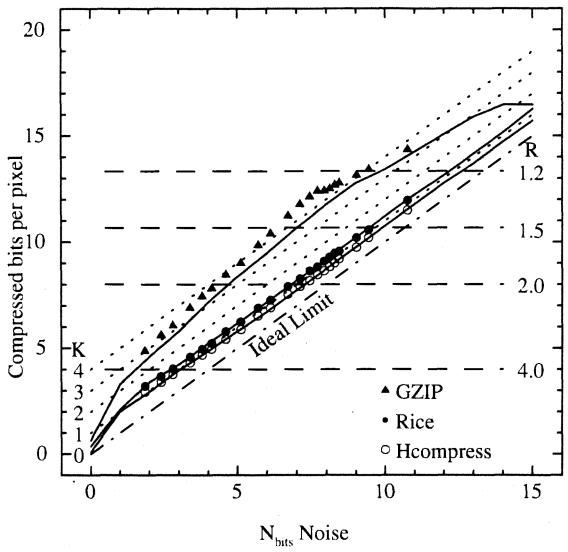

Fig. 1.— Plot of compressed bits per pixel versus the number of noise bits in 16-bit synthetic images. The solid lines represent the images that have N_{bits} of pure noise, and the symbols represent the images that have Gaussian distributed noise, where N_{bits} is calculated from Equation 4. The solid circles are for Rice compression, the open circle for Hcompress, and the triangle are for GZIP.

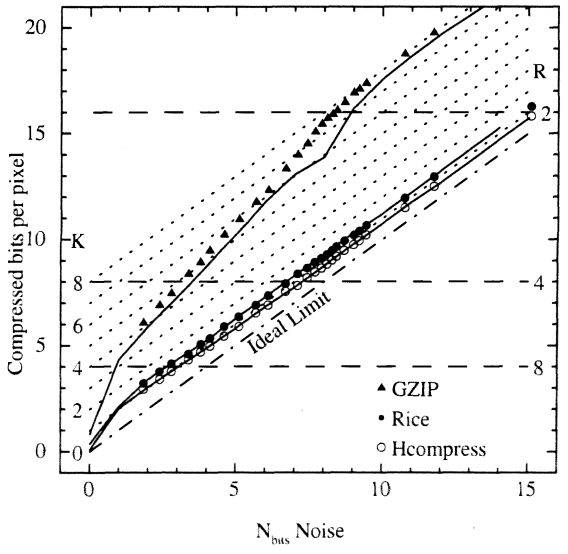

Fig. 2.— Same as Figure 1, except for 32-bit integer synthetic noise images.

a function of the number of actual or equivalent noise bits in the 16-bit images. This coordinate frame was chosen because lines with constant K have a slope of 1.0 and a Y-intercept = K, as shown by the dashed lines in the figure. The solid lines in the figure represent the images with N bits of pure noise, and the circles or triangles represent the other set of synthetic images with Gaussian-distributed noise, where the equivalent number of noise bits is calculated from Equation 4. The corresponding compression ratio, R, is shown by the horizontal dashed lines.

It can be seen in Figure 1 that the Rice and Hcompress algorithms do have the expected slope of 1, and they have constant K values, independent of the amount of noise in the image, of about 1.2 and 0.9, respectively. The close agreement between the lines (derived from the pure noise synthetic images) and corresponding points (derived from the Gaussian noise images) for these 2 compression algorithms confirms the validity of Equation 4 for computing the equivalent number of noise bits in images. Close inspection shows that there is a slight flattening of the slope of these relations for $1 < N_{bits} < 5$, which can be attributed mainly to the fact that there is a small amount of fixed disk space "overhead" required to store the compressed images in a FITS binary table structure, and this overhead becomes relatively more significant as the size of the compressed image decreases. (See also the discussion in the appendix of the non-linear behavior at small values of N).

It is also quite apparent that GZIP behaves very differently than Rice or Hcompress in Figure 1. GZIP cannot be parameterized with a single value of K, and instead it ranges from about 2 to 5, depending on the amount and distribution of the noise in the image. Unlike the other 2 algorithms, GZIP does not compress the 2 types of synthetic noise images equally well; it is more effective compressing the images in which the noise is confined to the lowest N bits. It is interesting that K appears to reach a maximum at $N_{bits} = 8$ which is where the noise propagates into the more significant byte of the 2-byte pixel values. This difference can probably be attributed to the fact that GZIP interprets each pixel as 2 independent 8-bit bytes whereas Rice and Hcompress treat each 16-bit pixel value as a single integer number.

The equivalent plot for the synthetic 32-bit in-

teger images is shown in Figure 2. The relations for Rice and Hcompress are virtually identical to those for 16-bit integers shown in Figure 1. and in particular, the K values are the same. This is expected because the size of the compressed image when using Rice or Hcompress only depends on the amount of noise, not on the byte size of the pixels. Thus if a 16-bit and 32-bit image have the same dimensions and have the same of amount of noise, the compressed files will be identical in size and hence the compression ratio of the 32-bit image will be exactly twice that of the the 16-bit image. As was the case with 16-bit images, GZIP behaves quite differently because of the way it interprets the image as a stream of bytes. The compression efficiency is much worse than Rice or Hcompress, with a variable K value that approaches a value of 8 for for the noisiest images.

Implicit in the above discussion is the fact that the compression ratio does not depend on the mean value of the pixels in the image; adding or subtracting a constant offset to all the pixels has no effect on the compression. Similarly, storing the image as signed or unsigned integers makes no difference. This is self-evident for the Rice and Hcompress algorithms since they operate only on the differences between pixels, not their absolute values. GZIP is also unaffected because the frequency distribution of different byte patterns is largely unchanged by applying a constant offset to the pixels.

4. Compression of 16-bit Astronomical Images

In contrast to the tests in the previous section using synthetic noise images, we now examine how well the different compression methods perform on actual 16-bit integer astronomical images. The primary data set used in these tests are the images that were taken during the night of 27 - 28 July 2006 at Cerro Tololo Inter-American Observatory using the Mosaic CCD camera. This camera contains 8 individual CCD detectors, and each detector has 2 amplifiers that read out half of the chip each. Every exposure with this camera results in a FITS file containing 16 image extensions that are each 1112 by 4096 pixels in size. In total, this data set consists of 102 FITS files containing 1632 separate FITS image extensions. To complement this large set of images taken with a single instrument, we also collected a small sample of other 16-bit integer images taken with other instruments that were available from various public astronomical data archives.

We compressed and uncompressed each of these images using the Rice, GZIP, and Hcompress algorithms supported by our tiled-image compression software, and in each case recorded the compression ratio, the calculated equivalent number of noise bits (from Equation 4), and the elapsed compression and uncompression CPU times. We also measured these same parameters when using the GZIP program on the host computer to externally compress and uncompress the images. These host GZIP tests were performed on a single FITS image extension instead of on the whole multi-extension file, to be comparable with the tiled-image compression tests which also operate on a single image extension at a time.

4.1. Compression Ratio versus Noise

One of most striking results of this study is shown in Figure 3, which plots the compression ratio using the Rice algorithm versus the measured number of equivalent noise bits in each image. The compression ratio R, is plotted, rather than the reciprocal quantity BITPIX/R that was use in the previous figures, because R is the quantity of more direct interest to most users.

The Mosaic camera CCD images (plotted with small + symbols) very closely follow the same curve as derived from the synthetic images that contain only noise (shown by the gray line). The somewhat surprising conclusion that can be drawn from this close agreement is that the compression ratio is almost solely dependent on the amount of background noise in the image. The actual image content, i.e., all the stars and galaxies seen in the images, has practically no influence on the compression ratio. The sample of 15 other CCD images taken with other instruments, as shown by the larger circles, also mainly follows the same curve. Only 2 of these points, plotted with open circles, have a significantly lower compression ratio than expected. Inspection of these 2 images showed that they contain such an unusually dense pattern of overlapping star images that it adversely affects the compression efficiency. In the great majority of cases, however, the image con-

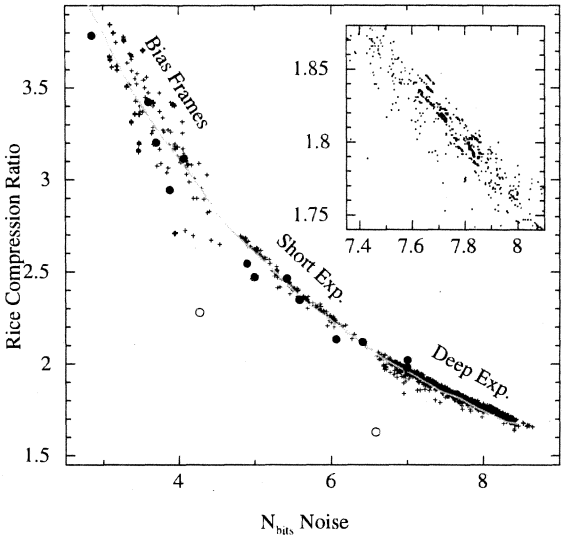

Fig. 3.— The Rice compression ratio as a function of the amount of noise in 16-bit integer images. The gray line is derived from the synthetic noise images.

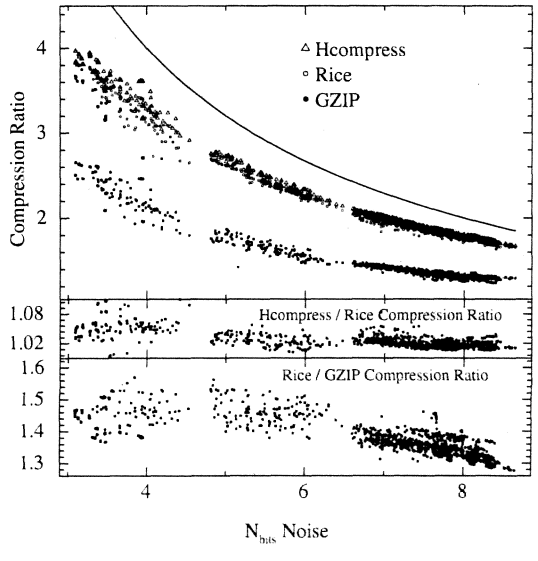

Fig. 4.— The compression ratios for the different algorithms as a function of the noise in the 16-bit images (upper panel). The solid line shows the theoretical upper limit for an ideal compression algorithm. The middle and lower panels compare the Rice ratio to that of Hcompress and GZIP, respectively.

tent has very little effect on the amount of lossless compression.

It is worth noting that much of the scatter for the Mosaic camera images seen in this figure is due to slight differences between the 16 image detectors. The insert in Figure 3 shows a magnified section of the data in which it can be seen that the points tend to lie along distinct bands that correspond to the different detectors. Each detector has unique characteristics in its fixed pattern noise which cause the noise in the pixel values to not be truly randomly distributed. This causes a slight systematical bias in the noise calculation (from Equation 1), producing small horizontal displacements of the points from each detector in the figure. Thus, most of the apparent scatter is the result of the superposition of the different detectors.

One other interesting thing to note in Figure 3 is that the different types of Mosaic camera images are segregated into different regions of the plot. The "bias" frame images that have 0 exposure time all have less than 4.5 equivalent bits of noise. The short calibration exposures occupy the middle region of the plot, with 4.5 to 6.5 bits of noise. Finally, the deep sky exposures, as well as the heavily exposed flat field images, have more than 6.5 bits of noise. This is a natural consequence of the fact that the different types of images have characteristically different mean count levels. Since the noise in a photon counting type detector scales as the square root of the number of detected counts, the different types of images will also have distinctly different noise levels. It is a somewhat perverse fact of nature that the more scientifically interesting astronomical images tend to have the most noise and hence have the worst compression factors.

4.2. Comparison of Different Compression Algorithms

Next we compare the file compression ratios and speed of the different compression methods. Figure 4 shows the compression ratios achieved by Rice, Hcompress, and tiled-GZIP plotted as a function of the equivalent number of noise bits for the 1632 Mosaic camera CCD images.

As can be seen, Rice and Hcompress achieve very similar compression ratios that are both much

higher than that produced by tiled-GZIP. The middle panel of the plot compares Rice and Hcompress, showing that Hcompress produces about 2% to 5% better compression. As discussed below, this small gain is usually not cost effective because of the much higher CPU times needed by Hcompress compared to Rice. The lower panel shows that Rice produces about 1.5 times better compression than tiled-GZIP for images with low to moderate amounts of noise and about 1.3 times better compression for the noisiest images.

The solid curve in the top panel of Figure 4 shows the theoretical maximum compression ratio, given by BITPIX / N_{bits}, that would be produced by an ideal algorithm that compresses all the non-noise bits in the image to zero size (i.e., an algorithm with $K = 0$ in Equation 3). By comparison, Rice and Hcompress have K values of about 1.2 and 0.9 bits per pixel, respectively, which means that they already achieve about 75% to 90% (depending on the noise level) of the theoretically best possible compression. Thus, it is not possible for any new lossless compression algorithm to produce dramatically better compression for astronomical data than is already achieved by Rice and Hcompress.

The relative compression and uncompression speeds[1] of the different methods are shown in Figures 5 and 6. When compressing images, Rice is 2 to 3 times faster than Hcompress, depending on the amount of noise, and 4 to 6 times faster than tiled-GZIP (or 2.5 to 3.4 times faster than host-GZIP). And when uncompressing images, Rice is 2 to 3 times faster than Hcompress, and 1.6 to 2 times faster than tiled-GZIP (or close to the same speed as host-GZIP).

The mean compression ratios and the relative compression and uncompression CPU times for the 1632 Mosaic camera images when using the different compression methods are summarized in Table 1, where the CPU times in each case are relative to the time when using the Rice algorithm.

[1]All the timing measurements in this paper are based on CPU times and not on the total elapsed processing times. In principle the elapsed time should be somewhat greater than the CPU time because it includes the latency times need to read or write data on magnetic disk. In practice, however, the sophisticated data caching systems on modern disks and computer systems tends to minimize these I/O bottlenecks, to the point where they are difficult measure consistently.

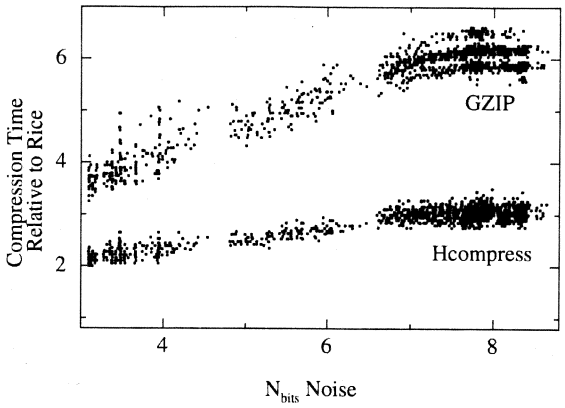

Fig. 5.— Relative CPU time needed to compress 16-bit integer FITS images using the GZIP (top) or Hcompress (bottom) algorithms as a function of the image noise level. The times are relative to the time when using the Rice algorithm. The horizontal banding of the points is due to the finite time resolution of the CPU measurements.

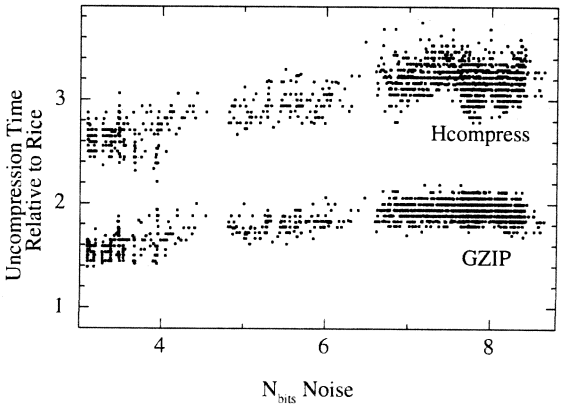

Fig. 6.— Relative CPU time needed to uncompress 16-bit integer FITS images using the GZIP (top) or Hcompress (bottom) algorithms as a function of the image noise level. The times are relative to the time when using the Rice algorithm. The horizontal banding of the points is due to the finite time resolution of the CPU measurements.

The uncompression speeds are arguably more important than the compression speeds because in many instances an image only has to be compressed once (by the data provider) but it must be uncompressed by every user, sometimes multiple times if the analysis software directly reads the image in its compressed form. The actual compression or uncompression speed depends of course on the underlying computing system. As a bench mark reference, one of our current Linux machines with a 2.4 GHz AMD Opteron 250 dual core processor (using only one of the processors) can tile Rice compress a 50 MB 16-bit integer FITS image in 1 second. Uncompressing this image also takes about 1 second.

4.3. Special case: Data Masks

Data mask images are often used in data analysis as a means of flagging special conditions that affect the corresponding pixels in an associated astronomical image. Typically, a small fraction of the pixels have one of a limited set of positive 16-bit integer values, and the remaining large majority of pixels are all equal to 0 or 1. These data masks have so little noise that they do not follow the general relationship between noise and compression ratio seen in more typical astronomical images. The compression ratio is not limited by the amount of noise, and instead is limited by the intrinsic internal overheads associated with each compression algorithm and with the FITS tiled-image file structure itself. With the Rice algorithm, for example, each block of 32 image pixels (= 64 bytes) can, in the extreme case where all the pixels have the same value, be compressed to a single 4-bit code value for a maximum compression ratio of 128. A similar analysis of the other available algorithms shows that the maximum compression ratio is about 200 for GZIP and about 700 for both Hcompress and PLIO. These maximum compression ratios are only achieved with relatively large image tiles sizes. For smaller tiles, less than a few thousand bytes in size, the compression algorithms become less efficient, and other fixed-size overheads, such as the 8-bytes per tile needed to store the byte offset and size of each tile in the FITS binary table format, become relatively more significant. Even so, the data masks still usually compress by a factor of 50 or more.

In many cases, achieving the very highest possible compression of data masks is of little practical benefit because the size of the compressed data mask image becomes insignificant compared to the rest of the associated data set. Each data mask image has an uncompressed FITS header that is at least a few thousand bytes in size, and each data mask is usually paired with an equal-sized astronomical image that itself can usually only be compressed by a factor of 2 - 3. Thus, the size of the compressed data mask is only a small percentage of the total data set size, and reducing the size even further makes very little practical difference. The compressed data masks are essentially 'free' for data providers because they take up almost no disk storage space compared to the rest of the data set.

Since the compression ratio is less of a factor in choosing which algorithm to use, the speed of the algorithm becomes a more significant consideration. Rice and PLIO are the fastest algorithms when compressing data masks, but GZIP and Hcompress are less than a factor of 2 slower, depending slightly on tile size. Overall, PLIO provides the best combination of compression ratio and speed when compressing data masks, but in practice it may be more convenient to use the same compression algorithm on the data mask as is used on the associated astronomical image.

5. Compression of 32-bit Integer Images

In order to measure the performance of the different compression methods on 32-bit integer images, we obtained a sample of FITS images taken with the NEWFIRM near-infrared camera during the night of 24 - 25 February 2008 at Kitt Peak National Observatory. The NEWFIRM instrument contains a mosaic of 4 imaging detectors, each of which is 2112 by 2048 pixels in size. There are 447 NEWFIRM observations in our data sample, giving a total of 1788 separate images.

5.1. Comparison of Different Compression Algorithms

We repeated the previous tests to measure the compression ratios and the CPU times required to compress and uncompress each of the 32-bit integer NEWFIRM images using the different compression methods. Figure 7 shows how the Rice and tiled-GZIP compression ratios vary as a func-

TABLE 1
16-BIT INTEGER IMAGE COMPRESSION

	Rice	Hcompress	Tiled-GZIP	Host-GZIP
Compression Ratio	2.11	2.18	1.53	1.64
Compression CPU time	1.0	2.8	5.6	2.6
Uncompression CPU time	1.0	3.1	1.9	0.85

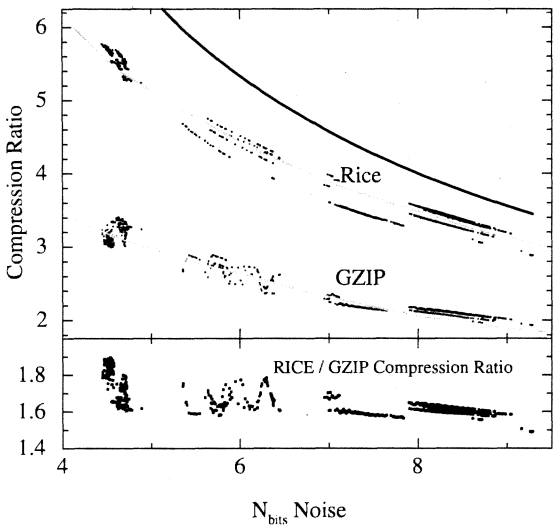

Fig. 7.— 32-bit integer image compression ratio for the Rice (upper points) and GZIP (lower points) algorithms plotted as a function of the noise level in the image. The analogous points for the Hcompress algorithm are not shown because they fall nearly on top of the Rice points. The gray lines going though these points are derived from synthetic images in which the pixel values have a constant level plus a known amount of Gaussian distributed noise. The points are segregated onto 4 distinct bands, which correspond to the 4 different detectors in the NEWFIRM mosaic camera.

tion of the measured number of equivalent noise bits in each image. (The points for the Hcompress algorithm have been omitted for clarity in this figure because they lie only slightly above the Rice points). This figure is similar to the corresponding Figure 4 for 16-bit integer images, except that the compression ratios are about twice as large, given the same amount of image noise. This is a natural result of the fact that a 32-bit integer image is twice as large as a corresponding 16-bit image, but the compressed size of the image, at least when using Rice or Hcompress, only depends on the noise, not on the intrinsic bit-length of the pixels. One important consequence of this fact is that there is no disk space penalty in storing FITS images as 32-bit integer arrays instead of 16-bit integer arrays, because the Rice or Hcompress compressed images are identical in size. It still does require slightly more CPU time to compress or uncompress the 32-bit representation of the image, however.

This 2:1 relationship in compression ratios does not hold for the GZIP algorithm because GZIP operates on the individual bytes in the image data, not on the numerical 2-byte or 4-byte integer pixel values. The presence of the 2 higher order bytes in the 32-bit image degrades the compression efficiency when using GZIP even if those 2 bytes are equal to zero in every pixel. Thus, the compression ratio of a 32-bit image when using GZIP is only about 1.6 times greater then that of a 16-bit image with the same noise level.

It can also be seen in Figure 7 that the points are segregated into 4 distinct bands that correspond to the 4 different detectors in the NEWFIRM mosaic camera. Unlike the CCD detectors in the MOSAIC camera, which are very closely matched in image quality and noise characteristics, the 4 infrared imaging devices in the custom-

built NEWFIRM camera have distinctive characteristics. In particular, one of the detectors appears to have significantly less noise than in the other 3 detectors (i.e., the points are shifted to the left). This is at least partially due to the fact that there is a systematic offset between the mean background level in the even and odd numbered columns in this detector. Since our noise estimation algorithm (see section 3) depends on the differences between every other pixel in each row, this under estimates the true pixel-to-pixel noise variation in the images taken with this detector.

As was the case for 16-bit images, the different types of images fall in different regions of Figure 7. The clump of points with $N_{bits} < 5$ and with Rice compression factors greater than 5 correspond to 'bias' calibration exposures that were taken with a closed shutter. The points with $5 < N_{bits} < 7$ and Rice compression factors of about 4.5 correspond to short exposures of calibration stars, and the remaining points with larger noise values and Rice compression factors of $\sim$ 3.5 correspond to the deep sky images as well as the heavily exposed flat field images.

For comparison, the upper solid curve in Figure 7 shows the maximum possible compression ratio that would be achieved by an ideal compression algorithm that has $K = 0$. Hcompress and Rice have K values of = 0.9 and 1.2, respectively, exactly the sames as with 16-bit integers, and thus are again within 70% to 90% of this theoretical limit, depending on the amount noise in the image.

Figures 8 and 9 compare the CPU times required to compress and uncompress the 32-bit integer images with GZIP or Hcompress, relative to the time required when using the Rice algorithm. These are analogous to Figures 5 and 6 for 16-bit images. The average compression ratios and the relative compression and uncompression CPU times for all 1788 images are also summarized in Table 2. As can be seen, the speed advantage of Rice over Hcompress or tiled-GZIP is even greater when compressing or uncompressing 32-bit images than with 16-bit images. Our bench mark Linux machine (2.4 GHz AMD Opteron 250 dual core processor), can Rice-compress a 90 MB 32-bit integer FITS image in about 1 second and can uncompress the same image in about 1.2 seconds.

Over all, Rice is clearly the best lossless com-

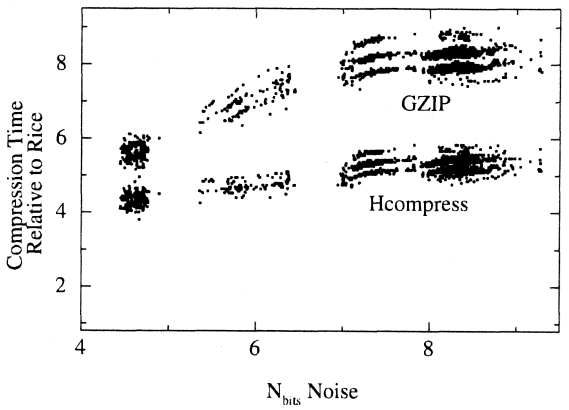

Fig. 8.— CPU time needed to compress 32-bit FITS images using the GZIP (top) or Hcompress (bottom) algorithms as a function of the image noise level. The times are relative to the time to compress the same image using the Rice algorithm.

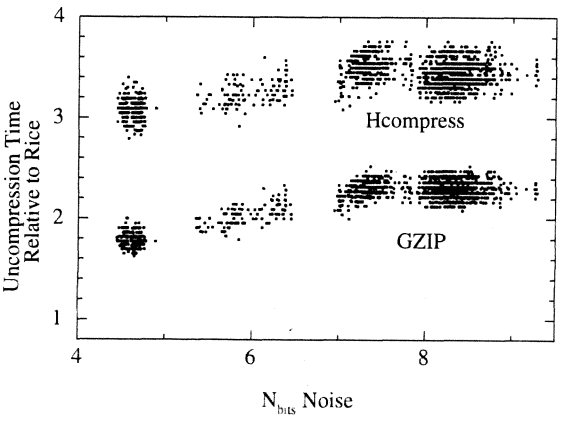

Fig. 9.— CPU time needed to uncompress 32-bit FITS images using the GZIP (top) or Hcompress (bottom) algorithms as a function of the image noise level. The times are relative to the time to compress the same image using the Rice algorithm.

TABLE 2
32-BIT INTEGER IMAGE COMPRESSION

	Rice	Hcompress	Tiled-GZIP	Host-GZIP
Compression Ratio	3.76	3.83	2.30	2.32
Compression CPU time	1.0	5.2	7.8	4.7
Uncompression CPU time	1.0	3.4	2.2	1.3

pression method for 32-bit integer images. In the case of a typical deep-sky image with 8 equivalent bits of noise, Rice produces 1.6 times better compression than tiled-GZIP, and has 8 times the compression speed and 2 times the uncompression speed. Compared with Hcompress, Rice is 5 and 3.5 times faster, respectively, when compressing and uncompressing images, which more than makes up for the slight 2% difference in compression ratio.

5.2. Special case: Representing Floating-Point Images as Scaled Integers

Instead of representing floating-point pixels directly in images, a widely used FITS convention converts the floating-point values into scaled integers, where the (approximate) floating point value is then given by

$$\mathsf{real\_value} = \mathsf{BSCALE} \times \mathsf{integer\_value} + \mathsf{BZERO} \quad (6)$$

and where BSCALE and BZERO are linear scaling constants given as keywords in the header of the FITS image. This is technically a 'lossy' compression technique that quantizes all the pixel values into a set of discrete levels, spaced at intervals of 1/BSCALE. Ideally, the quantization levels should be spaced finely enough so as to not lose any scientific information in the image, but without also preserving excessive amounts of noise.

Unfortunately, a common practice is to simply compute the BSCALE value so that the minimum and maximum pixel values are scaled to span the full 32-bit dynamic range of the scaled integer image. This has the effect of *magnifying* the noise in the floating-point array by a huge factor, which makes the scaled integer array virtually incompressible.

In order to achieve higher compression, a better technique, as first described by White & Greenfield (1999), is to choose the BSCALE value so that the quantized levels are spaced at some reasonably small fraction of the noise in the image, such that,

$$\mathsf{spacing} = 1/\mathsf{BSCALE} = \sigma/D \quad (7)$$

and where σ is calculated from Equation 1. The number of noise bits per pixel that are preserved in this case can be calculated from Equation 4 and is simply $\log_2(D) + 1.792$. In order to achieve the best compression, data providers should choose the smallest value of D that still preserves the required scientific information in the compressed image. This will depend on the particular application, but previous experiments (see for example Figure 2 in White & Greenfield, 1999) suggest that values of D in the range of 10 to 100 may be appropriate.

6. Compression of Floating-point Images

FITS images that have 32-bit floating-point pixel-values are more challenging to compress than integer images for 2 reasons. Firstly, many compression algorithms, including Rice and Hcompress, can by design only operate on integer data. Secondly, floating-point pixel values often contain a large amount of noise which greatly hinders compression. Since a 32-bit floating-point value can record about 6.5 decimal places of precision, whereas most astronomical images rarely have more than 3 decimal places of significance per pixel, this means that many of the lower order bits in each pixel value effectively just contain incompressible random noise. As a result, even algorithms like GZIP which can losslessly compress floating-point data (since it treats each byte as an independent datum) are not very effective. To test this, we collected a sample of 17 floating-point images from public astronomical archives taken with

different instruments. Of these, 13 of the images compressed very poorly with GZIP as expected, with low compression factors ranging from 1.08 to 1.25. Surprisingly, the other 4 images did quite well, with compression factors of 2 to more than 5. Closer inspection showed that these are not typical floating-point images: the pixel values in these images are quantized into a limited set of discrete levels and therefore effectively behave more like integer images in so far as compression is concerned.

Given these difficulties, it is generally not cost effective to losslessly compress floating-point FITS images. Therefore, we use the technique discussed in section 5.2 which converts the floating-point values into scaled 32-bit integers and then compresses them using the Rice (by default) algorithm. The linear scaling parameters are calculated independently for each tile of the image, and the BSCALE value is derived based on the amount of noise in each tile to preserve only a user-specified number of noise bits in the compressed image. By discarding the remaining noise bits, the image compression ratio can be dramatically increased. It is, however, incumbent upon the user to determine the appropriate number of noise bits to be preserved so as to not degrade the scientific usefulness of the image. As a rough guide, we have found that retaining 6 to 8 bits of noise in the scaled integer image is often sufficient. As can be seen from Figure 7, this will result in image compression ratios of about 4 when using the Rice algorithm.

7. Effect of Tiling Pattern on Compression Performance

There are a number considerations in choosing an appropriate tiling pattern when compressing an image. First, the tile must be sufficiently large for the compression algorithm to operate efficiently. For the Rice algorithm, the lower limit is about 500 pixels; for GZIP it is about 2000. Below these levels, the compression time for the image and the size of the compressed file both begin to increase. The Hcompress algorithm is inherently different from Rice and GZIP in that the wavelet transform only operates on 2-dimensional arrays of data. At a minimum it requires tiles containing at least 4 rows of the image, and it reaches near maximum efficiency when the tiles contain about 16 rows. For this reason we adopted 16 rows of the image at a time as the default tiling pattern in our software when using Hcompress.

The other main consideration when choosing a tile size is how the software that reads the image will access the pixels. The 2 most common access methods used by astronomical software are either to read the entire array of pixels in the image into computer memory all at once, or to read the image sequentially one row at a time. In the first case, the specific tiling pattern makes very little difference because the reading routine simply has to uncompress each and every tile in the image once and pass the array of uncompressed pixels back to the application program.

If the application program reads the image one row at a time, then the tiling pattern can have a major effect on the reading speed. If each tile contains multiple rows of the image (and in the limit, the whole image could be compressed as one big tile), then the FITS file reading routine has to uncompress the whole tile in order to extract just a single row. It would obviously be very inefficient to repeatedly uncompress the same tile each time the application program requests the next row of pixels. Instead, a recommended implementation strategy is to temporarily store the most recently accessed uncompressed tile in memory, so that it is immediately available in case the application program reads more pixels from that same tile. This caching technique adds some computational overhead, however, so in general the default single row tiling pattern is the most efficient for applications that read an image row by row.

A third type of image access occurs in applications that read a rectangular 'cutout' from a much larger compressed image. In this case it can be efficient to use a rectangular tile pattern that approximates the size of the typical cutout. Only those tiles that overlap the cutout region will then have to be uncompressed. This tiling pattern may be grossly inefficient however, for software that accesses the image one row at a time, unless a fairly sophisticated caching mechanism is implemented to store all the uncompressed tiles along a row.

In summary, the default row by row tiling pattern (or 16 rows at a time in the case of Hcompress) should work well in most situations. The main exception is if the images are very small, in which case it may be more efficient to compress multiple rows, or the entire image, as a tile.

8. Summary

In this paper we have demonstrated how the presence of random noise in an image almost completely determines how much the image can be losslessly compressed. The average number of noise bits per pixel in an image can be accurately derived from the Gaussian sigma of the pixel variations in background regions of the image. Since these random noise bits are inherently incompressible, the maximum possible lossless compression ratio, in the ideal case where all the remaining non-noise bits are compressed to zero, is simply given by the number of bits per pixel divided by the average number of noise bits per pixel. In practice of course, no actual compression algorithm can achieve this ideal amount of compression, and instead can only compress the non-noise bits down to some some finite value of K bits per pixel. The K value of each algorithm can be empirically measured and can be used to rank the compression efficiency of the algorithms. The most efficient algorithms used in this study are Hcompress and Rice, which have K values of 0.9 and 1.2 bits per pixel, respectively. When compressing a typical integer CCD image with 8 equivalent noise bits per pixel, these algorithms achieve close to 90% of the maximum possible amount of compression that would be achieved by a ideal algorithm with $K = 0$. The Rice algorithm in particular is also exceptionally fast, so it is deemed unlikely that any new algorithm that may be developed in the future will be able to match its combination of speed and compression efficiency.

We use a relatively new FITS format convention for storing compressed images in which the image is divided into rectangular tiles (usually row by row) and then each tile is compressed and stored in a row of a FITS binary table structure. The main advantages of this compression format over the common technique of externally compressing the whole FITS file with a file compression tool like GZIP are, (1) the FITS header keywords remain uncompressed for fast read and write access, (2) small sections of an image can be read without having to uncompress the whole image, and (3) a single image extension in a multi-extension FITS file can be read without uncompressing the entire file.

This tiled image compression technique offers a choice of different compression algorithms. The Rice and Hcompress algorithms are generally much more effective then GZIP at losslessly compressing astronomical images because they operate on the numerical 16 or 32 bit pixel values rather than just treating them as sequence of independent 8-bit bytes. As a result, Rice and Hcompress produce about 1.3 times more compression than GZIP with 16-bit integer images, and about 1.6 times more with 32-bit images. Although Rice and Hcompress produce similar amounts of compression, Rice is about 3 times faster and is therefore recommended for general use. Rice is also much faster than GZIP when compressing images and has about the same speed when uncompressing them.

We compared the various compression methods on a large sample of astronomical images obtained from NOAO as well as a smaller sample from other major observatories. Almost all these images followed the expected compression ratio versus noise relationship, with remarkably little scatter. This demonstrates that the actual content of the image, i.e., the stars and galaxies or other image features, have almost no effect on the compression ratio. We only found a few cases where the density of stellar images was so great that it adversely affected the amount of compression.

One interesting result from this comparison is the fact that the different types of astronomical images contain characteristically different amounts of noise and therefore have distinctly different compression ratios. The amount of noise correlates strongly with exposure time or the mean count level in the image, so the short exposure calibration images typically have less noise and compress better than deep images of the sky.

Finally, we investigated compression techniques for 32-bit floating-point astronomical images. Due to the large fraction of noise bits in these images, lossless compression is generally not cost effective. Instead, we developed a technique where the pixel values are converted to scaled 32-bit integers before compression. This is not a lossless compression technique, since quantizing the pixel values in this way discards some of the noise. When used properly, this technique will preserve the scientific integrity of the image, but by discarding some of the random noise it will give much higher compression ratios of about 4 or more, instead of only

about 1.2 when losslessly compressing the image with GZIP.

One of the main conclusions from this study is that the current lossless compression techniques (in particular, the tiled image Rice compression method) are very close to achieving the theoretical maximum limit that is set by the amount of noise in the images. Very little added benefit can be gained from developing better lossless compression algorithm. Instead, the only way to make significant improvements in astronomical image compression is to use lossy techniques which discard some of the incompressible noise without degrading the scientific information in the images.

The general purpose FITS tiled-image compression and uncompression software tools used in this study are publicly available from the HEASARC web site. These tools are distributed as part of the CFITSIO library package and can also be downloaded from a dedicated web page at the HEASARC web site. The programs are called *fpack* and *funpack* and are invoked on the command line to compress or uncompress an input list of FITS images, analogous to the gzip and gunzip utilities. Various user parameters can be specified on the command line to select which compression algorithm to use and to specify the desired image tile sizes. There is also a '-T' test option which can be used to generate a report which compares the compression ratio and speed of all available compression algorithms on the specified input images. More extensive information about using the fpack and funpack utility programs is available in the companion users guide that is included in the distribution.

A. Derivation of the Equivalent Number of Noise Bits

For images with a Gaussian noise distribution (for instance, the readout floor of a CCD), we derive the equivalent number of noise bits. Start by assuming N bits of uniform noise and average over the range of data numbers (x = DN) for the expected values of x and x^2:

$$\begin{aligned} \overline{x} &= [k(k+1)/2]/2^N \quad (\Sigma \text{ of DN series, with } k = 2^N - 1) \\ &= (2^N - 1)/2 \end{aligned}$$

and

$$\begin{aligned} \overline{x^2} &= [k(k+1)(2k+1)/6]/2^N \quad (\Sigma \text{ of series of squares}) \\ &= (2^N - 1)(2^{N+1} - 1)/6 \end{aligned}$$

Solve for the variance,

$$\begin{aligned} \sigma^2 &= \overline{x^2} - \overline{x}^2 \\ &= (2^{2N} - 1)/12 \end{aligned}$$

In the limit of large N:

$$\sigma = 2^N/\sqrt{12}$$

Solving for N then gives,

$$\begin{aligned} N_{bits} &= \log_2(\sigma\sqrt{12}) \\ &= \log_2(\sigma) + 1.792 \end{aligned}$$

The factor of $1/\sqrt{12}$ can be identified as the familiar analog-digital quantization noise (Janesick 2001). This can be derived with continuous variables by integrating the second moment of a stepwise probability density symmetrically over 2^N quanta. The discrete derivation above makes the non-linear limiting behavior at small values of N evident.

Figures 1 and 2 demonstrate empirical agreement of synthetic data with the N_{bits} relation for both 16-bit and 32-bit integer pixels. Figure 3 and 7 empirically confirm this relation for real world optical and infrared data sets.

REFERENCES

Gailly, J.L. & Adler, M, 1992, in RFC 1952, available from http://www.ietf.org/rfc/rfc1952.txt & http://www.gzip.org.

Hanisch, R. et al. 2001, A&A, 376, 359

Janesick, J. R. 2001, Scientific Charge-Coupled Devices (Bellingham, WA: SPIE Press)

Nieto-Santisteban, M. A., Fixsen, D. J., Offenberg, J. D., Hanisch, R. J. & Stockman, H. S. 1999, in ASP Conf. Ser., Vol. 172, Astronomical Data Analysis Software and Systems VIII, eds. D. M. Mehringer, R. L. Plante, & D. A. Roberts (San Francisco: ASP), 487

Pence, W. D. 1999, in ASP Conf. Ser., Vol. 172, Astronomical Data Analysis Software and Systems VIII, eds. D. M. Mehringer, R. L. Plante, & D. A. Roberts (San Francisco: ASP), 487

Pence, W. D., White, R. L., Greenfield, P., & Tody, D. 2000, in ASP Conf. Ser., Vol. 216, Astronomical Data Analysis Software and Systems IX, eds. N. Manset, C. Veillet, & D. Crabtree (San Francisco: ASP), 551

Percival, J. W. & White, R. L. 1996, in ASP Conf. Ser., Vol 101, Astronomical Data Analysis Software and Systems V, eds. G. H. Jacoby & J. Barnes (San Francisco: ASP), 108

Rice, R. F., Yeh, P.-S., & Miller, W. H. 1993, in Proc. of the 9th AIAA Computing in Aerospace Conf., (AIAA-93-4541-CP), American Institute of Aeronautics and Astronautics

Seaman, R., Pence, W. D., White, R., Dickinson, M., Valdes, F., & Zarate, N. 2007, in ASP Conf. Ser., Vol. 376, Astronomical Data Analysis Software and Systems XVI, eds. R. A. Shaw, R. Hill, & D. J. Bell (San Francisco: ASP), 483

Stoehr, F. et al. 2007, ST-ECF Newsletter. 42, 4

Tody, D. 1993, in ASP Conf. Ser., Vol 52, Astronomical Data Analysis Software and Systems II, eds. R.J. Hanisch, R.J.V. Brissenden, & J. Barnes (San Francisco: ASP), 173

White, R. L. & Becker, I. 1998, in SPIE Proc., Vol. 3356, Space Telescopes and Instruments V, eds. P. Y. Bely & J. B. Breckinridge (Bellingham: SPIE), 823

White, R. & Greenfield, P. 1999, in ASP Conf. Ser., Vol. 172, ADASS VIII, eds. D. M. Mehringer, R. L. Plante, & D. A. Roberts (San Francisco: ASP), 125

White, R., Postman, M. & Lattanzi, M. 1992, in Digitized Optical Sky Surveys, eds, H. T. MacGillivray and E.B. ThompsonKluwer, Dordrecht, 167

This 2-column preprint was prepared with the AAS LaTeX macros v5.2.